GYM MANAGEMENT SYSTEM

GROUP MEMBERS:

1. KINGSUK PANDA 23CS8086

2. ARKAPRAVA DHAR 23CS8087

3. RAAYAN HALDER 23CS8088

4. SANGRAM BASAK 23CS8089

5. SHADAB FAHIM 23CS8090

PROJECT DETAILS:

THIS PROJECT IS CREATED AS A PART OF THE DATABASE MANAGEMENT SYSTEMS COURSE TO DEVELOP AND IMPLEMENT A WORKING RELATIONAL DATABASE FOR A GYM. IT DEALS WITH HANDLING MEMBER RECORDS, TRAINER ALLOTMENTS, AND SESSION TRACKING USING DISCIPLINED DATA MODELS AND SQL OPERATIONS. THE SYSTEM MAINTAINS DATA CONSISTENCY, UPHOLDS INTEGRITY CONSTRAINTS, AND GIVES MEANINGFUL RESULTS THROUGH OPTIMIZED QUERIES.

GITHUB REPOSITORY LINK:

https://github.com/ArkaD-23/DBMS\_Project

# **Table of Contents**

1. Introduction
2. Objective
3. System Design
4. Entity-Relationship Diagram
5. Database Schema
6. Stored Procedures
7. Triggers
8. Operational Queries
9. Data Insertion
10. Conclusion

# **Introduction**

In modern fitness centers, managing member records, trainer allocations, and session tracking manually leads to inefficiency and inconsistency. This project presents a comprehensive SQL-based backend system to digitize and optimize gym management. The proposed solution automates member enrollment, session logging, trainer management, and insightful reporting through structured queries, stored procedures, triggers, and indexing.

# **Objective**

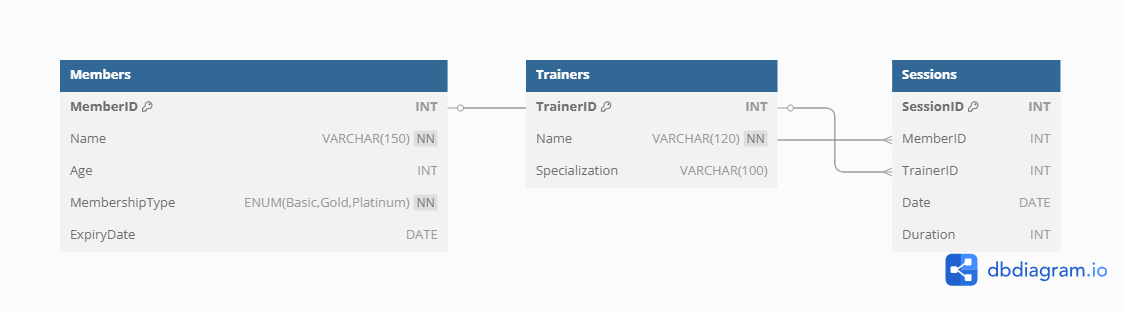
- Design and implement a relational database for a gym.  
 - Track members, trainers, and their sessions efficiently.  
 - Provide analytics like busiest days, top trainers, and session counts.  
 - Implement business rules using constraints, procedures, and triggers.  
 - Maintain data integrity and automate audit trails.

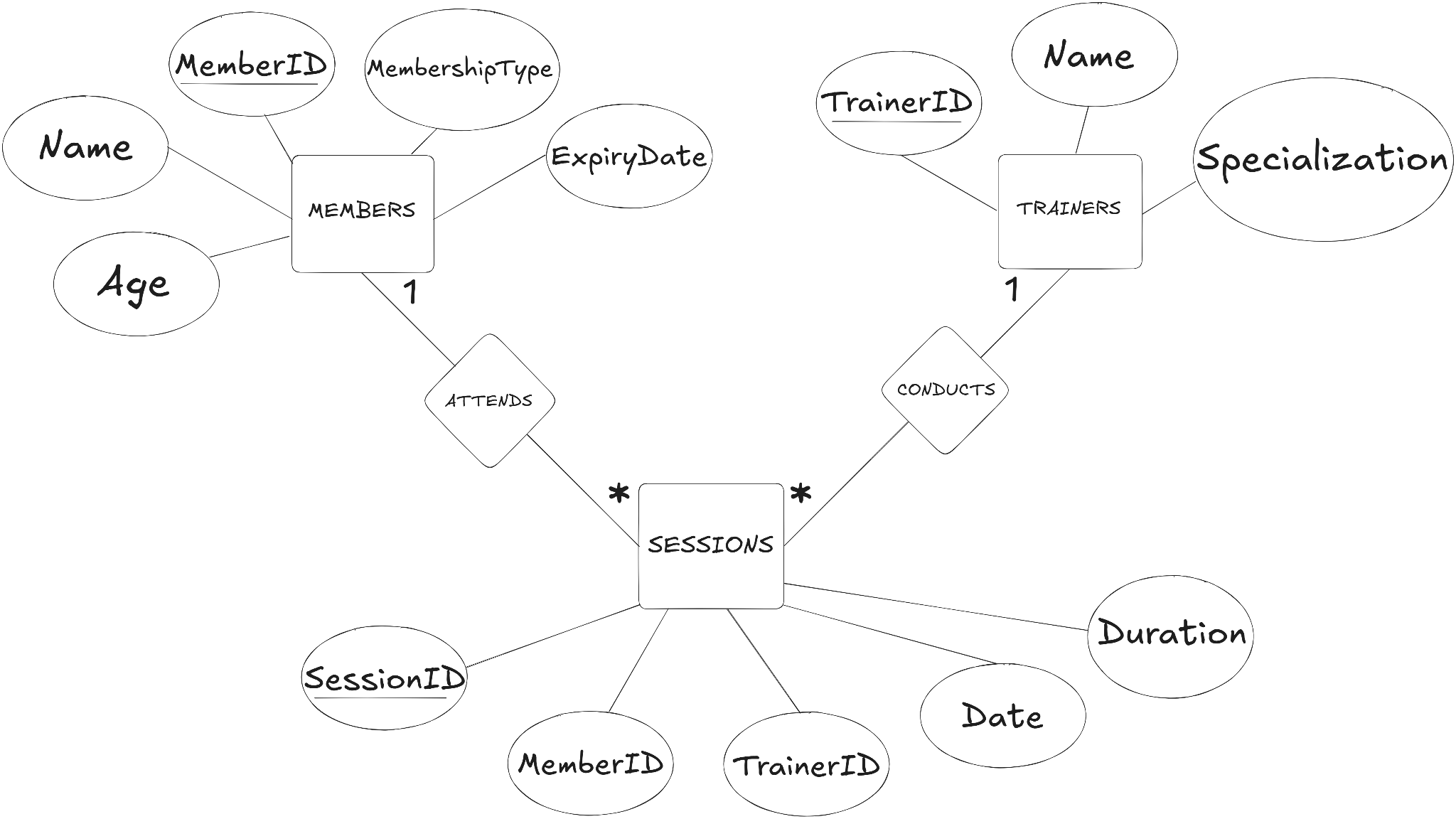
# **System Design**

The system revolves around three core entities:  
 - Members: Individuals with memberships.  
 - Trainers: Fitness professionals with specialization.  
 - Sessions: Booked workouts connecting members and trainers.  
  
 The design enforces relationships and constraints to ensure data validity and supports real-time analytics for decision-making.

# 

# **Entity-Relationship Diagram**





# **Database Schema**

Tables include Members, Trainers, and Sessions, each enforcing key constraints for integrity.

**Members (MemberID, Name, Age, MembershipType,ExpiryDate)**

**Trainers (TrainerID, Name, Specialization)**

**Sessions (SessionID, MemberID, TrainerID, Date, Duration)**

# 

# **Stored Procedures**

Examples include procedures to fetch session data, extend memberships, update types, and calculate session durations.

# **Triggers**

Implemented for:  
 - Logging changes in membership expiry.  
 - Preventing expired members from booking sessions.

# **Operational Queries**

SQL queries for analytics: session count, top trainers, attendance, duration summaries, and more.

# 

# 

# 

# 

# **Data Insertion**

## **SQL Stored Procedure Report – Membership System**

### **🔹 File Name: stored\_procedures.sql**

### **🔹 Purpose:**

This file defines several stored procedures used for **retrieving analytics**, **managing memberships**, and **generating useful insights** from the members, sessions, and trainers tables.

### **GetSessionsByMember**

#### **Procedure Name: GetSessionsByMember(IN member\_id INT)**

#### **Purpose:**

Fetches all session records attended by a specific member.

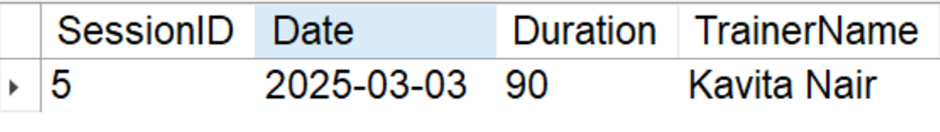
#### 

#### **Logic:**

* Joins sessions and trainers to retrieve:  
  + SessionID, Date, Duration, and TrainerName
* Filters sessions by the provided member\_id.

#### **Example Usage:**

CALL GetSessionsByMember(2);



### **ExtendMembership**

#### **Procedure Name: ExtendMembership(IN member\_id INT, IN extra\_days INT)**

#### **Purpose:**

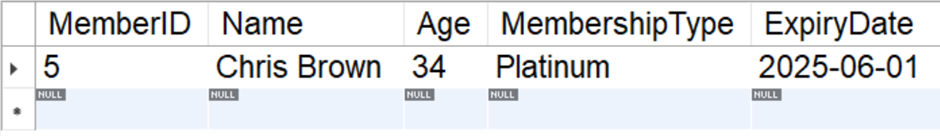
Adds a specified number of days to a member’s current membership expiry.

#### **Logic:**

* Uses DATE\_ADD to extend ExpiryDate by extra\_days.

#### **Example Usage:**

CALL ExtendMembership(5, 31);



### **GetActiveNonAttendees**

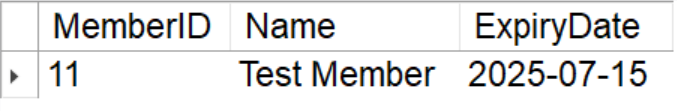
### **Procedure Name: GetActiveNonAttendees()**

#### **Purpose:**

Retrieves active members who **have never attended any session**.

#### **Logic:**

* Checks members with ExpiryDate >= CURDATE().
* Excludes those with session records in the sessions table using NOT IN.



### **UpdateMembershipType**

#### **Procedure Name: UpdateMembershipType(IN m\_id INT, IN new\_type ENUM('Basic', 'Gold', 'Platinum'))**

#### **Purpose:**

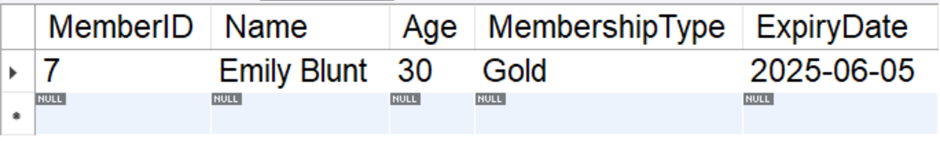
Updates the membership type of a specific member.

#### **Logic:**

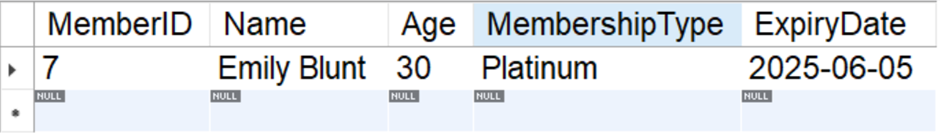
* Sets MembershipType to the new value where MemberID matches.

#### **Example Usage:**

CALL UpdateMembershipType(7, 'Platinum');  
OLD



NEW



### **GetTotalWorkoutDuration**

#### **Procedure Name: GetTotalWorkoutDuration(IN m\_id INT)**

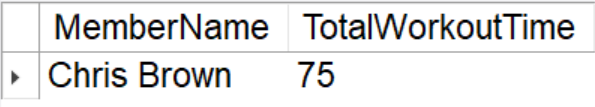
#### **Purpose:**

Calculates the total workout time for a member across all sessions.

#### **Logic:**

* Joins members and sessions.
* Uses SUM(Duration) and groups by member.

CALL GetTotalWorkoutDuration(5);



### **TrainerSessionCount**

#### **Procedure Name: TrainerSessionCount()**

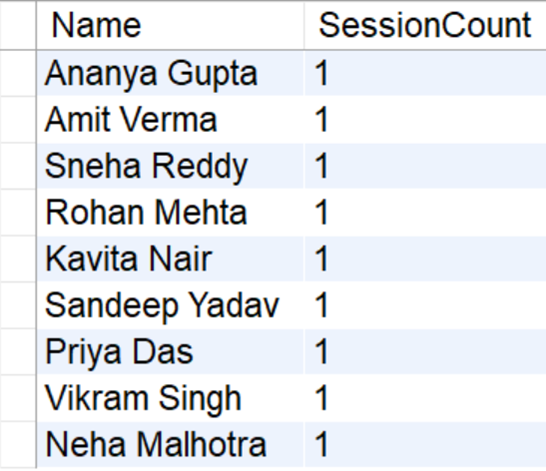
#### **Purpose:**

Lists all trainers with the number of sessions they’ve conducted.

#### **Logic:**

* Performs a LEFT JOIN to count sessions per trainer (even those with none).
* Groups by TrainerID.

CALL TrainerSessionCount()



## **Summary Table:**

| **Procedure** | **Benefit** |
| --- | --- |
| GetSessionsByMember | Track individual attendance |
| ExtendMembership | Support for subscription management |
| GetActiveNonAttendees | Identify under-utilized memberships |
| UpdateMembershipType | Dynamic member upgrade/downgrade |
| GetTotalWorkoutDuration | Insights into member workout consistency |
| TrainerSessionCount | Trainer performance analysis |

## 

## 

## 

## **SQL Operations Report – queries.sql**

### **File: queries.sql**

### **Purpose:**

The file contains SQL operations designed for retrieving, analyzing, and updating data related to gym members, trainers, sessions, and membership statuses.

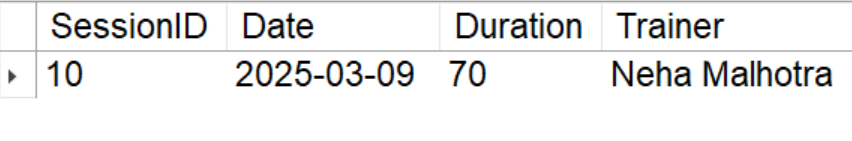
**1. Sessions Attended by a Specific Member**

SELECT SS.SessionID, SS.Date, SS.Duration, T.Name AS Trainer

FROM Sessions SS

JOIN Trainers T ON SS.TrainerID = T.TrainerID

WHERE SS.MemberID = 4;



### **2. List All Trainers and Their Specializations**

SELECT TrainerID, Name, Specialization

FROM trainers;



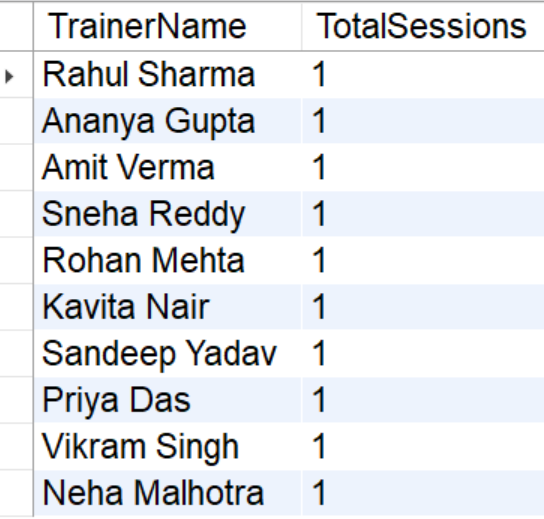
### **3. Number of Sessions per Trainer**

SELECT T.Name AS TrainerName, COUNT(DISTINCT SS.SessionID) AS TotalSessions

FROM trainers T

JOIN sessions SS ON T.TrainerID = SS.TrainerID

GROUP BY T.TrainerID, T.Name;



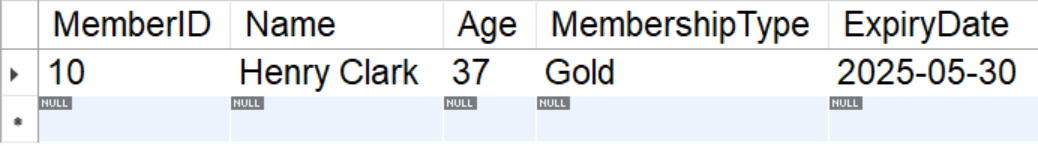
### **4. Update a Member’s Membership Type**

UPDATE members

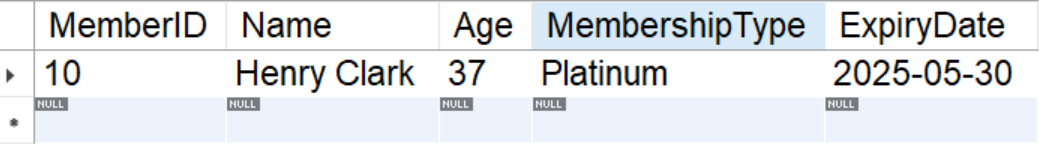
SET MembershipType = 'Platinum'

WHERE MemberID = 10;

OLD



NEW



### **5. Total Workout Duration for a Member**

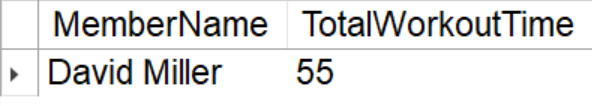
SELECT M.Name as MemberName, SUM(SS.Duration) AS TotalWorkoutTime

FROM Members M

JOIN Sessions SS ON M.MemberID = SS.MemberID

WHERE M.MemberID = 6

GROUP BY M.MemberID, M.Name;



### **6. Most Frequently Attended Trainer**

SELECT T.Name AS MostFrequentTrainer, COUNT(\*) AS SessionCount

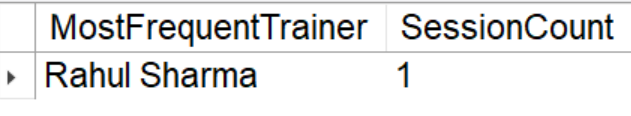
FROM Sessions SS

JOIN Trainers T ON SS.TrainerID = T.TrainerID

GROUP BY T.TrainerID, T.Name

ORDER BY SessionCount DESC

LIMIT 1;

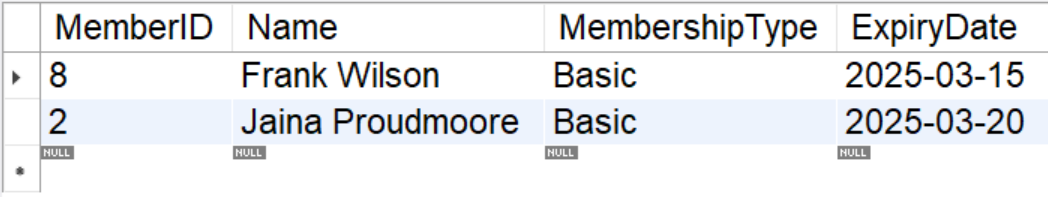


### **7. Members with Expired Memberships**

SELECT MemberID, Name, MembershipType, ExpiryDate

FROM members

WHERE ExpiryDate < CURDATE();



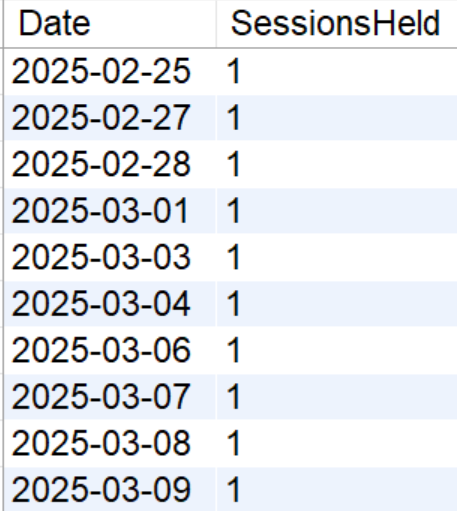
### **8. Busiest Training Days**

SELECT Date, COUNT(\*) AS SessionsHeld

FROM sessions

GROUP BY Date

ORDER BY SessionsHeld DESC;



### **9. Members with More Than 10 Sessions**

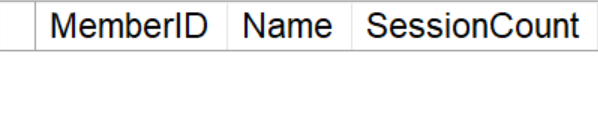
SELECT M.MemberID, M.Name, COUNT(SS.SessionID) AS SessionCount

FROM members M

JOIN sessions SS ON M.MemberID = SS.MemberID

GROUP BY M.MemberID, M.Name

HAVING COUNT(SS.SessionID) > 10;

****

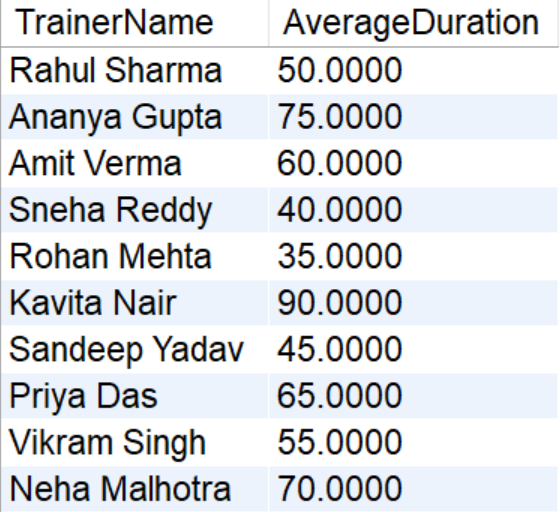
### **10. Average Session Duration per Trainer**

SELECT T.Name AS TrainerName, AVG(SS.Duration) AS AverageDuration

FROM trainers T

JOIN sessions SS ON T.TrainerID = SS.TrainerID

GROUP BY T.TrainerID, T.Name;



## **Summary Table**

| **#** | **Query Purpose** | **Key Tables Involved** | **Output Highlights** |
| --- | --- | --- | --- |
| 1 | Sessions attended by a member | sessions, trainers | Session details for Member 8 |
| 2 | List all trainers | trainers | Names and specializations |
| 3 | Sessions count per trainer | trainers, sessions | Session counts grouped by name |
| 4 | Update membership type | members | Sets type to Platinum |
| 5 | Total workout time | members, sessions | Total duration for Member 1 |
| 6 | Most frequent trainer | trainers, sessions | Top trainer by sessions |
| 7 | Members with expired memberships | members | Shows expired users |
| 8 | Busiest training days | sessions | Dates with highest activity |
| 9 | Members with >10 sessions | members, sessions | Loyalty or high-usage members |
| 10 | Average session duration per trainer | trainers, sessions | Performance or planning insights |

## **SQL Trigger Report – Membership System**

### **File Name: triggers.sql**

### **Purpose:**

This script defines SQL triggers to **audit membership expiry changes** and **enforce business rules** that prevent session bookings for members with expired memberships.

### **1. Table: membership\_audit**

#### **Purpose:**

Logs any changes made to the expiry date of a member’s subscription.

#### **Table Structure:**

| **Column** | **Data Type** | **Description** |
| --- | --- | --- |
| AuditID | INT | Auto-increment primary key |
| MemberID | INT | References the member whose expiry changed |
| OldExpiry | DATE | Previous expiry date of the membership |
| NewExpiry | DATE | Updated expiry date |
| ChangeDate | TIMESTAMP | Date and time when the change was logged |

### 

### **2. Trigger: trg\_log\_expiry\_update**

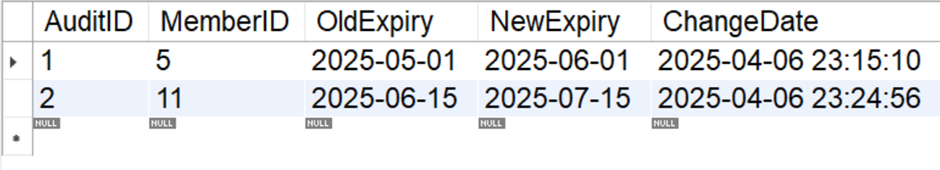
#### **Trigger Type: BEFORE UPDATE ON members**

#### **Purpose:**

Automatically logs changes in the ExpiryDate field of the members table to the membership\_audit table.

#### **Logic:**

* Trigger fires **before an update** occurs on the members table.
* If the ExpiryDate is changed (OLD.ExpiryDate <> NEW.ExpiryDate), it logs:  
  + MemberID
  + OldExpiry
  + NewExpiry
  + ChangeDate



### **3. Trigger: trg\_prevent\_expired\_sessions**

#### **Trigger Type: BEFORE INSERT ON sessions**

#### **Purpose:**

Prevents booking a session for a member **whose membership has already expired**.

#### **Logic:**

* Trigger fires **before a new session is inserted**.
* It checks the ExpiryDate of the MemberID involved.

If the membership expiry is **before the current date**, it blocks the insert using:  
  
 SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Cannot book session: Membership expired';

INSERT INTO sessions (MemberID, TrainerID, Date, Duration)

VALUES (2, 1, '2025-04-07', 60);



## **SQL Data Insertion Summary – insert\_data.sql**

### **Purpose:**

Populate the database with sample data for realistic testing and reporting across core entities: members, trainers, and sessions.

### **1. Members Table**

INSERT INTO Members (Name, Age, MembershipType, ExpiryDate)

VALUES

('John Smith', 26, 'Gold', '2025-06-15'),

('Jaina Proudmoore', 30, 'Basic', '2025-03-20'),

('Alice Johnson', 25, 'Platinum', '2025-07-10'),

('Bob Odenkirk', 43, 'Gold', '2025-06-30'),

('Chris Brown', 34, 'Basic', '2025-05-01'),

('David Miller', 28, 'Platinum', '2025-08-25'),

('Emily Blunt', 30, 'Gold', '2025-06-05'),

('Frank Wilson', 32, 'Basic', '2025-03-15'),

('Grace Hall', 29, 'Platinum', '2025-07-20'),

('Henry Clark', 37, 'Gold', '2025-05-30');

**OUTPUT:**



### **2. Trainers Table**

INSERT INTO Trainers (Name, Specialization)

VALUES

('Rahul Sharma', 'Strength Training'),

('Ananya Gupta', 'Yoga & Flexibility'),

('Amit Verma', 'Weight Loss & Cardio'),

('Sneha Reddy', 'Pilates & Core Strength'),

('Rohan Mehta', 'Functional Fitness'),

('Kavita Nair', 'CrossFit'),

('Sandeep Yadav', 'Powerlifting'),

('Priya Das', 'Zumba & Aerobics'),

('Vikram Singh', 'Martial Arts & Self-defense'),

('Neha Malhotra', 'Sports Conditioning');

**OUTPUT:**

****

### 

### **3. Sessions Table**

INSERT INTO Sessions (MemberID, TrainerID, Date, Duration)

VALUES

(3, 1, '2025-02-25', 50),

(1, 4, '2025-02-27', 40),

(5, 2, '2025-02-28', 75),

(7, 3, '2025-03-01', 60),

(2, 6, '2025-03-03', 90),

(8, 7, '2025-03-04', 45),

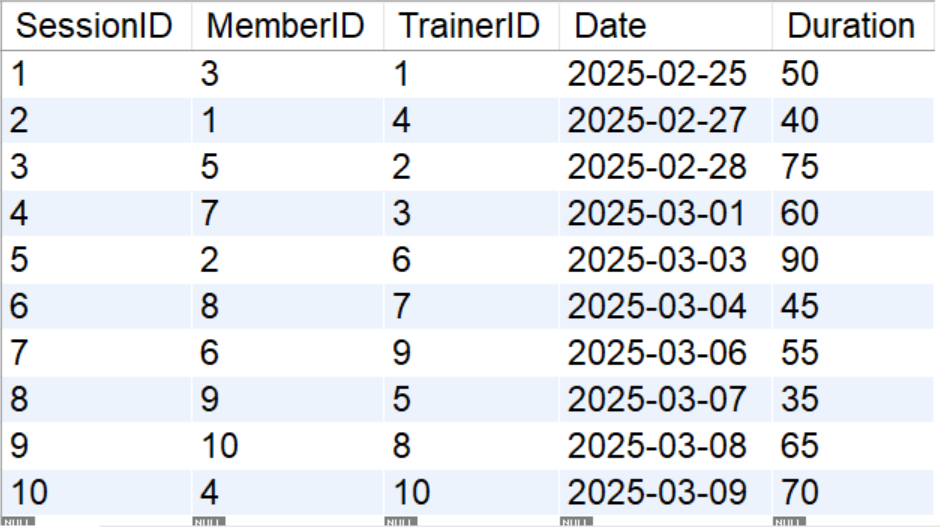
(6, 9, '2025-03-06', 55),

(9, 5, '2025-03-07', 35),

(10, 8, '2025-03-08', 65),

(4, 10, '2025-03-09', 70);

**OUTPUT:**



## 

## 

## **Data Insertion Validation Checklist**

| **Table** | **Rows Inserted** | **Key Fields** | **Notes** |
| --- | --- | --- | --- |
| Members | 10 | Name, Age, MembershipType | Mix of membership types for testing expiry |
| Trainers | 10 | Name, Specialization | Broad variety of fitness specializations |
| Sessions | 10 | MemberID, TrainerID, Duration | Covers diverse session durations & dates |

# **Conclusion**

The Gym Management System offers a complete backend solution using SQL. It ensures data integrity, supports administrative analytics, and is ready to integrate with front-end systems.